Fusion Engineering and Design 85 (2010) 419–422

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

Web services usage in distributed file systems

V.F. Pais[a,b,*]

[a] National Institute for Laser, Plasma and Radiation Physics (INFLPR), Lasers Department, Romania
[b] Faculty of Automatic Control and Computers, "Politehnica" University of Bucharest, Romania

ARTICLE INFO

Article history:
Available online 4 March 2010

Keywords:
Web service
Distributed file system

ABSTRACT

This paper investigates the possibility of using web services as the building blocks for distributed file system, instead of the traditional RPC or RMI.

1. Introduction

Fusion experiments are a complex research activity involving a large volume of data collected from each experiment. Processing of this data is usually accomplished by using parallel codes, running on multiple nodes of a cluster or in a GRID environment. Thus a distributed file system accessible from all the processing nodes is required. Furthermore, given the various hardware and software platforms used in fusion research, such a distributed file system cannot be linked to any specific operating system, like Microsoft DFS, available on some Microsoft Windows platforms, or OpenAFS available on Linux, but only partially supported on Windows.

Web services allow applications to interact without human intervention through dynamic connections. This automated interaction is possible due to technologies like Extensible Markup Language (XML), used for data transfer, XML Schema, for describing the data, and Web Service Definition Language (WSDL), for identifying the available web service methods.

Modern programming languages provide the required mechanisms for easily producing and consuming web services. In most cases, it is possible to have a "proxy" class automatically generated from the WSDL, thus becoming very easy to develop applications based on web calls.

Given the easiness associated with web service development, during the last years their usage in applications increased. However, this is true mostly for high level applications. Therefore, this paper tries to study the possibility of using web services for low level applications, especially a distributed file system.

In order to study the impact of having a web service interface offering distributed storage, a test file system was implemented: "OpenWebDFS".

2. OpenWebDFS nodes

Each machine taking part in this distributed file system is called a node. It can implement one or more of these functions: master, storage and access.

"Master" nodes offer locking mechanisms, while "storage" nodes are actually storing blocks of data. Each file can be found on several "storage" nodes and at least one "master" node can resolve concurrent access issues.

"Access" nodes offer an interface to the file system. They can take the form of a web interface or an operating system driver.

Considering the three types of nodes, a complete installation must contain at least one of each node type. In order to increase the available space, additional "storage" nodes can be added. Also, additional "master" nodes ensure protection against failure while working on certain files and additional "access" nodes allow file system access from various points.

OpenWebDFS functions were implemented using a combination of PHP and C. The high level language PHP was used to create the web services that provide most of the file system functions. In addition to the standard PHP functions, the NuSOAP [1] library was used. This choice is justified by its extensive debugging facilities, offering easy access to all parts of the SOAP messages (request/result/fault), thus making it easier to find and resolve errors.

An example "access node" has two parts: a web service implemented in PHP, offering methods for accessing files and directories and a FUSE (FileSystem in User Space) [2] Linux module, allowing the distributed file system to be mounted as a regular Linux file system.

* Correspondence address: National Institute for Laser, Plasma and Radiation Physics (INFLPR), Lasers Department, Atomistilor 409, PO Box MG-36, 077125 Magurele, Romania. Tel.: +40 721678357.
E-mail address: vasile.pais@inflpr.ro.

0920-3796/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.fusengdes.2010.02.013

The communication between the FUSE module and the PHP web service is achieved through the gSOAP [3] framework. gSOAP is a cross-platform open source C and C++ software development toolkit that generates C/C++ RPC code, XML data bindings, and efficient schema-specific parsers for SOAP Web services and other applications that benefit from an XML interface. The gSOAP toolkit offers a comprehensive and transparent XML data binding solution for C and C++ through autocoding techniques. Autocoding saves developers substantial time to implement SOAP/XML Web services in C/C++. In addition, the use of XML data bindings significantly simplifies the use of XML in applications by automatically mapping XML to C/C++ data types. Application developers no longer need to adjust the application logic to specific libraries and XML-centric data representations such as DOM.

The "storage node" is used to store the actual content of the various file system resources. The size reported by the file system is the total size of all storage nodes. Each OpenWebDFS implementation requires at least one storage node.

Data is stored in blocks. The size of these blocks must be the same across the entire file system. For each non-empty resource, at least one block will be allocated. In case the file size is not a multiple of block size, space will become wasted on the storage node. Thus, the reported used percentage of the file system may differ from the actual size of all files.

A "master node" stores information about the overall file system organization and status. However, the most important role of a master node is to provide locking mechanisms for accessing resources.

3. OpenWebDFS resources

In order to better understand the capabilities of OpenWebDFS, it is important to have a knowledge of how resources are represented internally. Additionally, some information regarding the sizes of the various elements involved is presented only to give a general idea of the additional data carried with each request between the file system nodes.

Everything is considered to be a "resource" (including directories, files and metadata). A "resource" has three characteristics: resource identifier (RID), resource map and (optional) storage blocks on one or more storage nodes.

The resource identifier is used for locating resources, therefore it must be unique and currently has a fixed length of 80 characters. The file/directory identified by a path name in a regular file system is translated by the access node to a RID. In order to facilitate access to certain resources, the first character in the RID denotes the type of resource (D = directory, B = binary, M = metadata, R = root).

When a master or storage node receives a request for an unknown resource it will respond by throwing a web service fault. This is propagated up the request chain to the access node that made the corresponding request. Finally, it is up to the access node to translate the fault into a file system error and give it to the operating system.

Whenever a new computer becomes a file system node, it will generate for itself a unique identifier. Currently it has a fixed length of 40 characters and is intended to allow for computers to have more than one or varying network addresses. This identifier then becomes part of all RIDs generated by the current server.

The actual content of a resource is stored on storage nodes and it is determined based on the resource type.

A resource map is created for each resource, containing the size of the resource content and its distribution across storage nodes. All resource maps are stored on the master node(s). They are not stored in the storage nodes and their size is not reflected in the used blocks of the file system. The structure of a resource map is presented in Fig. 1.

```
SIZE
MID,DID;MID,DID;.....
MID,DID;MID,DID;.....
.......
MID,DID;MID,DID;....
```

Fig. 1. Resource map structure.

The first line of the resource map always contains the resource true size (in bytes). If the size is 0, then this may be the only line the resource map (this is the case for a newly created resource). Starting with the second line, there are the addresses of the actual storage blocks used to hold the resource content. MID stands for Machine_ID of the storage node holding the block, while DID stands for Data_ID and is significant only for the storage node. It is the address of the storage block inside its storage node.

If a file is not replicated, then each line will contain a single MID,DID definition. For a replicated file, each block can be replicated on any number of machines (it can be replicated even multiple times on the same storage node). Since replication takes place at block level, rather then file level, each line can have a different number of MID,DID pairs. This is useful for creating a "replicator" daemon that takes care of replicating files in the background.

A "directory" resource holds links to other resources available in the current folder. It contains a RID on each line. An empty directory has an empty content (no RIDs are present). For each directory resource there is an associated metadata resource.

The "root" resource is a special "directory" resource, containing the base of the file system. There can be only one "root" resource for a certain OpenWebDFS implementation.

A "binary" resource is used to hold the content of regular files. The actual content of a file is stored in this resource "as-is" (without any modification from OpenWebDFS).

Metadata resources are used for storing information about other resources. Both "binary" and "directory" resources have one "metadata" resource that is automatically created and managed by the file system. Currently, such resources contain information regarding the creation and modification time, the rights and the name of the associated resource. There is no limitation on the characters that can be used for a resource name. Nevertheless, the operating system may impose certain restrictions for the name of a file/directory. Therefore, it is the user's responsibility to choose appropriate names when resources are to be accessed from multiple operating systems.

Each request between file system nodes must contain enough information to properly identify the requested resource. Additionally, during certain requests metadata must be read, for example when verifying the user rights or when resolving RIDs back to human readable names. Therefore, each message passed between two nodes carries additional information.

4. File system tests

In order to test the influence of web services over the file system performance, a small test network was created. It is comprised of two virtual machines, created using VMWare [4] Server 2.0, each having 1 64-bit CPU, 256 Mb RAM and 80 Gb HDD. They were running 64-bit Scientific Linux [5] with kernel 2.6.18-92.1.17.

The tests that were performed on this system were designed to detect the increase in network load due to encapsulation of data in web service specific messages. Since all the traffic is directed to the standard HTTP port 80, it was considered sufficient to analyze